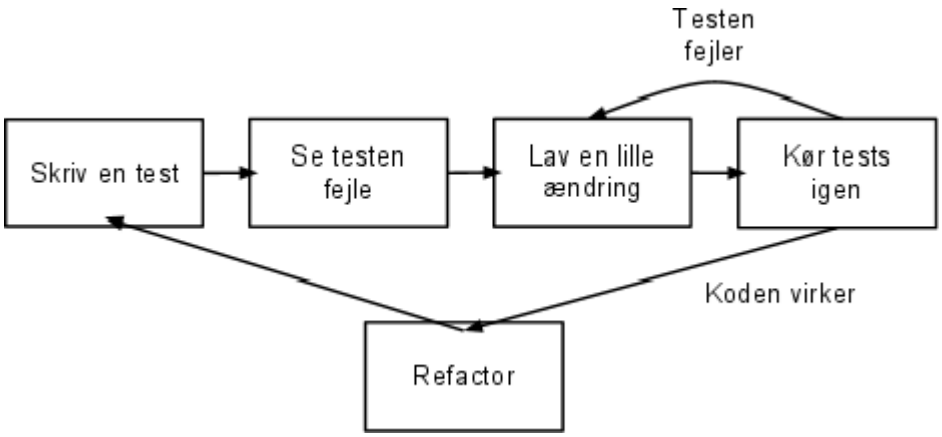
SWT Eksamen 2011

Disposition

- TDD
- Stubs
- Mocks

TDD

- Introduktion: formål, tankegang
    - Test Driven Development går ud på at vende udviklingsprocessen om og sikre at der bliver skrevet tests til al ens kode.
    - I stedet for at starte med implementationen, som man derefter tester, så starter man med at skrive en test, som man derefter skriver kode der sikrer går igennem. Dette sikrer at man udvikler et testkatalog til ens kode, i stedet for at man skal udvikle en masse tests senere, som formentlig aldrig dækker lige så meget.
    - Black-box testing: Man tester typisk kun public metoder. Dette gør også, at man internt kan refaktorere en klasse, uden at nogle tests breaker.
- Arbejdsgang



- Forskellige syn på TDD mht. design:
    - Test Driven Development: Design up-front (fra start)
    - Test Driven Design: Intet design up-front, testene skal drive designet
- Fordele
    - Deler processen op i mindre dele
    - Driver udviklingen fremad (hurtigt)
    - Bygger en stor testbase op omkring koden ⇛ kan give sikkerhed i større omskrivninger af koden
    - Hurtigt feedback - selvtillid
    - Ingen driverkode - xUnit sørger for det

    - Dokumentation - man kan se på tests hvad koden kan
    - Kun ønsket opførsel programmeres
    - Færre fejl i det endelige produkt
    - Fejl fanges tidligt i processen ⇒færre fejl senere ved integration test
- Ulemper
    - Hvis man ændrer noget fundementalt i produktionskode (ændrer interface) så skal man skrive alle tests om (man skal være sikker på sit design, inden man begynder på TDD)
    - TDD fanger ikke boundary conditions - kræver at man selv tænker over det
    - Andre tools kan være gode - f.eks. coverage tests til at sikre at man har dækket alle branches af koden
    - Fejlen kan godt ligge i testkoden - Skriv ALTID korte overskuelige tests
        i. undgå loops, logik, branches, osv. i tests

Stubs

- Formål
    - Erstatter eksterne afhængigheder - typiske ikke-deterministiske objekter (fx et terningekast, der ikke skal være tilfældigt)
    - Man ved præcist, hvad stubben gør, da man laver en (midlertidig) implementation
    - Man tester ikke forløbet, men sluttilstanden på UUT
- **Seam:** Steder, hvor man kan sætte en stub ind - trækkes typisk ud i interfaces, således man nemt kan erstatte et rigtigt objekt med en stub.
- Fordele
    - Skaber sikkerhed i forhold til at bruge ekstern kode/kilder
    - Kræver ikke ekstra framework
- Ulemper
    - Kræver meget boiler-plate kode - uinteressant kode

Mocks

- Formål
    - Undgå at skrive en implementation, som kun skal bruges til testen
    - Verificering af processen, f.eks. at et metodekald i en klasse følger en foruddefineret sekvens (tester opførsel)
- Detaljer
    - Strict: Fejler hvis en uventet metode kaldes, en forventet metode ikke kaldes, en forventet exception ikke smides, eller hvis der smides en uventet exception.
    - Nonstrict: Fejler kun, hvis en forventet metode ikke kaldes.
    - Recording: Man formulerer det forventede forløb, som et metodekald sætter igang, og verificerer, at dette rent faktisk sker.
    - Parametre: Man kan kontrollere (sætte begrænsninger) at input til parametre er korrekt.
    - Arrange-act-assert: Vi behøver ikke at recorde, hvad vi forventer, der skal ske.

Bruger lambda-syntax. Man deler sin test kode op i 3 dele:
    i. Arranger objekter.
    ii. Kald metoder på objekter.
    iii. Assert på mocken.
- Fordele
    - Sparer boiler-plate kode
    - Sikrer at hele forløbet er korrekt, hvor man med stubs kun er interesseret i slutreturværdien
- Ulemper
    - Introducerer meget kode, både i frameworket, men også til opsætning, kan give problemer med overskueligheden af tests.
    - I forhold til TDD kan det sætte tempoet ned, da testene typisk ikke vil være korte.
    - Ikke altid læsevenlige tests
    - Hvis der ændres for meget internt i en metode, der testes, kan en mock-test, hvor et metodekaldsforløb verificeres pludselig breakes - hvis rækkefølgen af kaldene fx ændres.

Mocks vs. stubs

- Test på Mock ⇔ test på UUT
- Stubbe verificerer resultatet ⇔ Mocks verificerer processen
- Nogen gange kan man ikke verificere tilstand (fx cache), her ville man bruge en mock
- Med mocks tænker man på den endelige implementation af klasser - det gør man ikke med stubs. Dette mener nogen er forkert set ud fra TDD-synspunkt, hvor man først skal tænke på implementation efter testen er skrevet.
- Eksempelkode
    - Record-and-replay.
    - Is.Anything kunne skiftes ud med Is.TypeOf for at tjekke, at det er den rigtige Transport-protokol, der bruges.